Matrix Multiplication

1 Introduzione

L'*assignment* prevede il calcolo del prodotto matriciale tra due matrici di dimensione $N \times N$ mediante varie strategie:

- CPU
- GPU tramite *global memory*
- GPU tramite *shared memory*
- Più GPU

La *shared memory* si stima essere circa 100 volte più veloce della *global memory* per cui è lecito aspettarsi un notevole miglioramento utilizzando tale tecnica. Si è scelto di sfruttare la *shared memory* anche nell'implementazione che sfrutta più GPU.

1.1 Moltiplicazione a blocchi

Per le due implementazione che sfruttano la *shared memory* si è utilizzata la moltiplicazione matriciale a blocchi. La matrice viene divisa in blocchi di dimensione32×32 in modo che ogni blocco possa essere memorizzato nella memoria condivisa da un blocco di thread della stessa dimensione e computato separatamente. La GPU *Tesla K40c* utilizzata per le prove mette a disposizione $49152byte$ per blocco, assicurando la possibilità di sfruttare blocchi della dimensione scelta, in quanto:

$$(32)^2 \cdot sizeof(int) = 32768byte < 49152byte$$

Anche la scheda video *GeForce GTX 750* che viene impiegata nella versione con due GPU ha la stessa dimensione di memoria condivisa per blocco, per cui non è stato necessario rivalutare la strategia.

2 Parallelizzazione

La moltiplicazione è di per sé un'operazione che permette di calcolare ogni singolo elemento della matrice in modo indipendente dagli altri, almeno utilizzando l'algoritmo più semplice. Per cui nella versione che usa la GPU

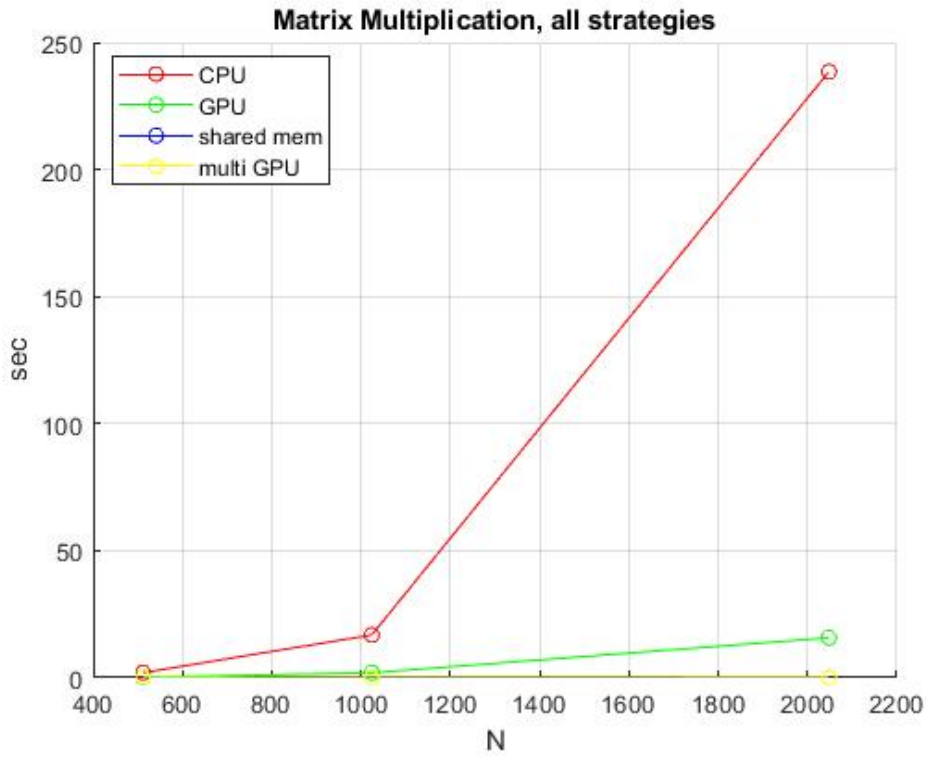
per rendere il calcolo parallelo usando la memoria globale è stato sufficiente inizializzare $N \times N$ blocchi da 1 thread ciascuno per eseguire il calcolo.

Le due versioni con memoria condivisa hanno invece necessitato la creazione di $N/32 \times N/32$ blocchi con 32×32 thread ciascuno. Ogni blocco esegue il compito di calcolare gli elementi appartenenti allo stesso blocco della matrice risultante.

Infine, per quanto riguarda la versione con due GPU si è divisa in due la matrice del *moltiplicatore* e del *prodotto*, in tal modo è stato sufficiente passare una singola metà di entrambe le matrici a ciascuna GPU. La matrice del *moltiplicando* è stata invece passata ad entrambe le GPU, poiché per computare mezzo prodotto è necessario conoscerla completamente.

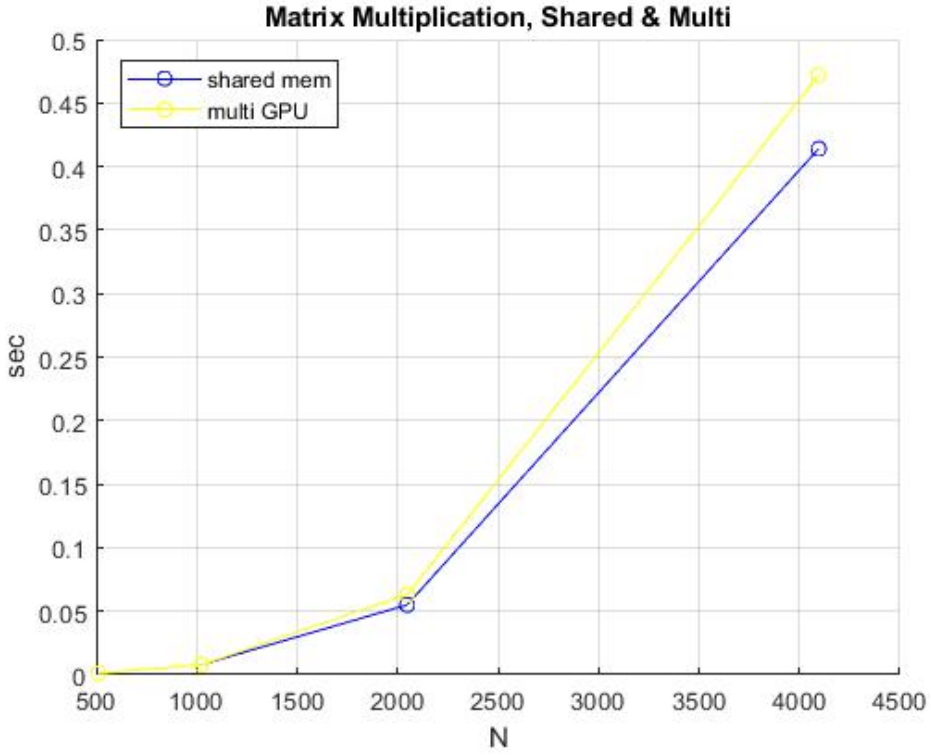
3 Benchmark

Si presenta ora il grafico relativo a tutti i test eseguiti sulle varie implementazioni dell'algoritmo, al crescere di N:



Si nota come la versione che sfrutta la CPU risulti nettamente meno performante delle versioni che sfruttano la GPU. Un ulteriore salto di qualità è visibile con l'ausilio della *shared memory*.

Al fine di evidenziare le differenza tra l'implementazione con una e due GPU si può visionare il seguente grafico che le compara separatamente:



La versione con singola GPU risulta essere leggermente più performante della versione con due GPU, questo è giustificato dal fatto che la *GeForce GTX 750* abbia prestazioni leggermente inferiori e il programma deve attendere che entrambe le schede video abbiano terminato la computazione prima di terminare.

4 Conclusioni

É stato possibile notare come la *shared memory* impatti sensibilmente sulle performance è some l'utilizzo di due GPU non è necessariamente vantaggio-

so. Con due GPU si possono ottenere, tuttavia, vantaggi nel caso in cui la matrice fosse di dimensione molto elevata tale da dover richiedere più cicli di computazioni causa il limitato numero di blocchi della GPU.